Logic for Computer Science

Project Statement

Pascal Fontaine, Tom Clara

October 2025

1 Presentation of the problem

In this project, you will use a SAT solver to solve the knight's tour problem. This problem consists in visiting exactly once each cell of a chessboard with a knight, starting at a given position. According to the rules of chess, a knight can move in (at most) 8 different ways on a chessboard, as depicted in Figure 1. Note that when the knight is close to an edge of the chessboard, the number of possible moves is lower, since it is obviously not allowed for the knight to move outside the chessboard. An example of solution for an 8×8 chessboard is shown

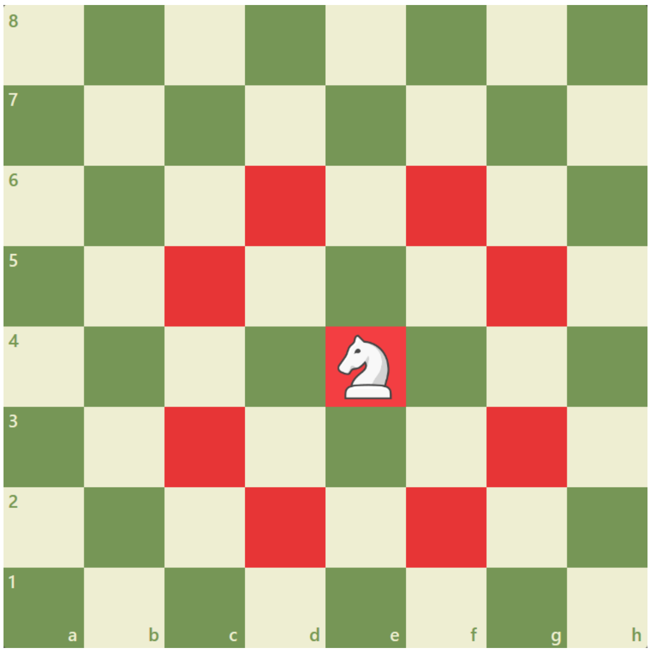
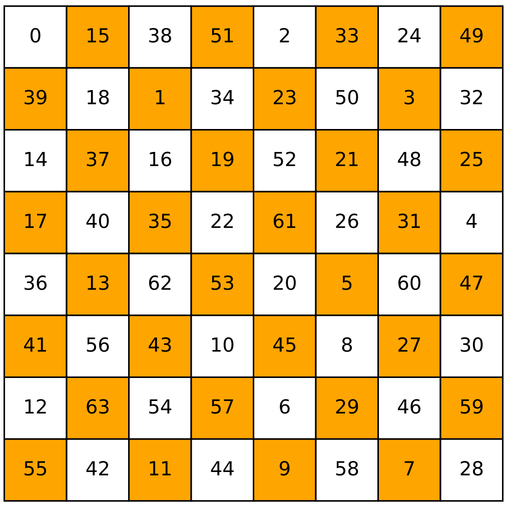


Figure 1: Moves allowed for a knight.



0	15	38	51	2	33	24	49
39	18	1	34	23	50	3	32
14	37	16	19	52	21	48	25
17	40	35	22	61	26	31	4
36	13	62	53	20	5	60	47
41	56	43	10	45	8	27	30
12	63	54	57	6	29	46	59
55	42	11	44	9	58	7	28

Figure 2: Example of solution, where the knight starts in the top left corner.

in Figure 2. In this example, the knight starts in the top left cell of the chessboard, labelled by 0. It then moves to the cell labelled by 1, and keeps moving until it reaches the final cell, labelled by 63. Each cell is visited exactly once, hence Figure 2 indeed shows a solution to the problem. If you want to play this game "by hand" for different chessboard sizes, please visit this page. You will quickly realize that the problem is hard to solve without a computer.

2 Instructions

General instructions

The project must be done individually. Your task will be to translate the problem into SAT and to explore some of its variations. Implement your solution in a python file (**use the template that is provided**). Section 3 contains the questions that must be answered. In addition to the python file, submit a brief pdf report (at most 2 pages) that explains, for each question, how you solved it using logic. This report should help us to understand your reasoning.

About the code

A template `solution_template.py` is provided. For each question, fill the body of the corresponding function, that has already been created for you. **Do not change the signature of these functions.** In other words, do not change the inputs, the outputs and the names of these functions. Moreover, do not print anything on standard output (this might interfere with the test script). Note that you can of course create other functions to structure your code.

It is not allowed to use external code, except the `Glucose3` solver from the `pysat.solvers` library. If you have a very good (technical) reason of not using this solver, please contact us as soon as possible.

Your code will be tested automatically. We provide a test script called `test-script.py`. Run it to test your implementation. This script does **not** check that your solution is entirely correct, but it might help finding some bugs.

Deadline and project assessment

Submit a zip archive containing the pdf report and the file `solution_template.py` containing the code. Please name this archive `sXXXXXX.zip`, where `sXXXXXX` is your student ID. Submit this on e-Campus before November 30th.

After the deadline, we will organize a live session where you will have to explain your code to us, and implement small modifications. We will check that you understand perfectly the code you submitted. The final grade will depend on:

- the score for the automatic tests,
- the global quality of the code and the report,
- and your ability to demonstrate that you understand everything you submitted.

3 Questions

1. Let M be the number of rows of the chessboard and N the number of columns. Let (i_0, j_0) be the initial position of the knight (where $0 \leq i_0 < M$ is the index of the row and $0 \leq j_0 < N$ is the index of the column). The function `question1` must solve the problem for a chessboard of size $M \times N$, and for a knight starting in cell (i_0, j_0). It should output a tuple $(\text{solution}, \text{solver}, \text{variables})$, where:

   - "solution" is a list of lists, such that $\text{solution}[i][j]$ is the number of moves required to arrive in cell (i, j). For instance, $\text{solution}[i_0][j_0] = 0$. If the problem cannot be solved (for instance, for a 4×4 chessboard), you must set $\text{solution}[i][j] = -1$ for all positions (i, j).
   - "solver" is the instance of the solver you used.
   - "variables" is the list of variables you used.

Note that outputting the solver and the variables allows you to reuse the function later, to solve variations of the problem.

Your function should be able to compute a solution to the problem (or detect the absence of solution) for all chessboards such that $1 \leq M, N \leq 7$, and for any initial position.

2. There are different ways of encoding the fact that the knight must visit each cell exactly once. Find an efficient encoding (and explain in the report the different options you tried). Then use it to modify the code of `question1`. This function should now be efficient enough to solve the problem for an 8×8 chessboard in a few seconds.[1]

   Hint: it is sometimes a good idea to add redundant clauses to quickly "guide" the SAT solver towards a solution.

3. In this question and in the next one, we set $M = 3$ and $N = 4$. Compute the number of solutions for this chessboard. The initial position of the knight is not imposed here. The function `question3` should output the number of solutions you computed (as an integer).

4. A 3×4 chessboard has three symmetries:

   - Two axial symmetries, whose respective axes are the vertical and horizontal blue lines in Figure 3.
   - A central symmetry, whose centre is the intersection of the blue lines in Figure 3.

   Count the number of solutions up to symmetry. In other words, you should now consider that two solutions are identical if one can be obtained by applying a symmetry to the other. For instance, the solutions represented in Figure 4 should be considered as identical. The function `question4` must compute and return the number of solutions (as an integer), up to symmetry.

5. In this last question, we would like to guarantee that the problem has a unique solution, by means of additional constraints. The function `question5` should have the same inputs as `question1`, and output a list of constraints, i.e., a list of the form $[(t_1, i_1, j_1), (t_2, i_2, j_2), ...]$ where the constraint (t, i, j) enforces the knight to visit cell (i, j) after exactly t moves. This list of constraints should guarantee that there is a unique solution S, and all constraints must be necessary for uniqueness of the solution (i.e., there must exist several solutions if any constraint is removed from the list). It the function is called with values M, N, i_0, j_0 such that no solution exists, it should return an empty list.

[1] Our solution takes 4 seconds on a very slow computer.

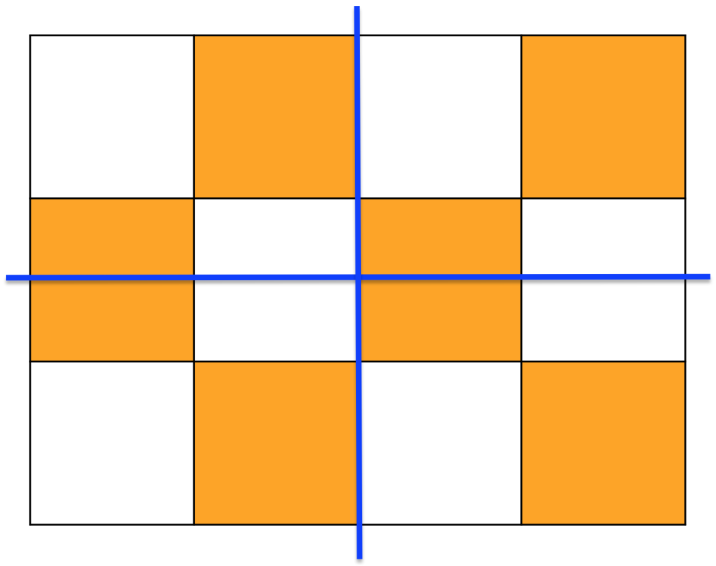
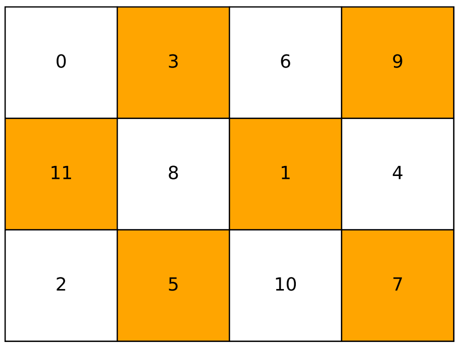
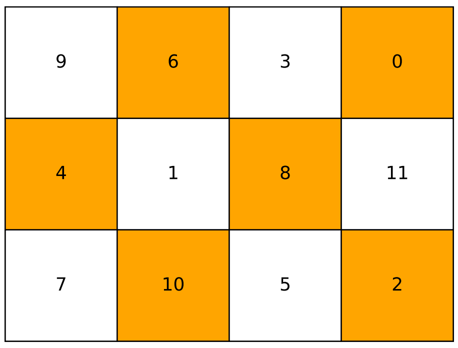


Figure 3: The symmetries of the 3×4 chessboard.



(a) First solution.



(b) Second solution, obtained by symmetry.

Figure 4: Two solutions considered identical.

Moreover, your function should be fair, in the sense that the constraints it generates should not always lead to the same solution S. You can use the library `random` if you need random numbers. Your function is expected to run reasonably quickly when $1 \leq M, N \leq 6$, and for any initial position of the knight.